MODULO IMPERATIVO

Una **lista** es una estructura de datos lineal compuesta por nodos. Cada nodo de la lista posee el dato que almacena la lista y la dirección del siguiente nodo. Toda lista puede recorrerse a partir de tu primer elemento. Los elementos no necesariamente están en posiciones contiguas de memoria. Para almacenar un nuevo elemento debe reservarse memoria dinámica y si se desea permite eliminar un elemento por medio de las operaciones *new* y *dispose*. Es *homogénea*, *dinámica* y *lineal*.

Operaciones permitidas: crear lista vacía, agregar elemento adelante o detrás, insertar un elemento, eliminar elemento, recorrer estructura, buscar elemento.

Un **arreglo** es una estructura de datos compuesta que permite acceder a cada componente por una variable índice, que da la posición del componente dentro de la estructura de datos. La estructura arreglo se almacena en posiciones contiguas de memoria. Es *homogénea, estática e indexada*. La cual maneja *dimF* y *dimL*.

Operaciones permitidas: cargar, agregar elemento, insertar elemento, eliminar elemento, recorrer estructura, buscar un elemento, ordenar la estructura.

ALGORITMO DE ORDENACIÓN: proceso por el cual un conjunto de elementos puede ser ordenado. Hay diferentes métodos de ordenación: **selección**, **intercambio** e **inserción**.

Cada uno tiene sus ventajas y desventajas en cuanto a facilidad de implementación, tiempo y memoria utilizada.

Método de inserción *p 225*

El ordenamiento por inserción (insertion sort) es una manera muy natural de ordenar un conjunto de elementos almacenados en un arreglo. Inicialmente se considera un solo elemento, que obviamente es un conjunto ordenado. Después, cuanod hay K elementos ordenados de menor a mayor, se toma el elemento K+1 y se comprara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se inserta el elemento L+1 debiendo desplazar los demás elementos. Requiere $O(n^2)$ operaciones para ordenar una lista de n elementos.

```
for i:= 2 to dimL
    actual:= v[i];
    j:= i-1;
    while (j > 0) and (v[j] > actual)
        v[j+1]:= v[j];
        j:= j -1;
    v[j+1]:= actual;
```

Método q ordena el vc de entrada insertando cada elemento a[i] entre los i-1 anteriores ya ordenados.

1 Método de inserción

Si los datos están ordenados de menos a mayor, el algoritmo solo hace comparaciones, por lo tanto, es de orden (n).

Si los datos están ordenados de mayor a menor el algoritmo hace todas las comparaciones y todos los intercambios, por lo tanto, el orden es n^2 comparaciones.

Supongamos **C**: nro. Comparaciones **M**: nro. Intercambios

$n-1 <= C <= n(n-1)/2 \quad 2(n-1) <= M <= 2(n-1)+n(n-1)/2$

Recursión

Metodología para resolver problemas. Permite resolver un problema P por resolución de instancias más pequeñas P1, P2, Pn del mismo problema

El problema Pi es de la misma naturaleza que el problema original, pero en algún sentido mas simple.

El problema es siempre el mismo, pero debe ir achicándose. Siempre tiene al menos un caso base, en el cual el código a implementar no es recursivo (a veces no debe escribirse código en el caso base, pero siempre existe). El caso base es el que determina el final de la solución recursiva.

Ejemplo

SOLUCION ITERATIVA

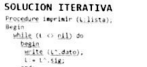
```
Procedure imprimir (L:lista);
Begin
  while (L <> nil) do
    begin
      write (L^.dato);
      L:= L^.sig;
    end;
End;
```

SOLUCION RECURSIVA

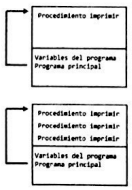
```
Procedure imprimir (L:lista);
Begin
  IF (L <> nil) do
    begin
      write (L^.dato);
      L:= L^.sig;
      imprimir (L);
    end;
End;
```

Qué pasa en la memoria?

SOLUCION ITERATIVA



SOLUCION RECURSIVA

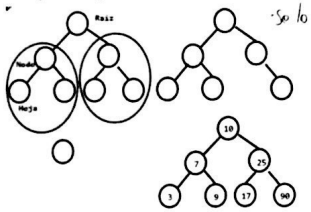


Supongamos una lista con tres elementos

Árbol binario de búsqueda

Es una estructura de datos jerárquica. Formada por nodos, donde cada nodo tiene a lo sumo dos hijos y mantienen un orden. Es *homogénea, dinámica, no lineal* y de *acceso secuencial.*

Apropiada p/la representación jerárquica




```
Programa arboles;

Type
  arbol = ^nodo;

  nodo = record
   dato: tipo;
   HI: arbol;
   HD: arbol;
  end;
```

O (log n)

CREACIÓN DE UN ARBOL

Hay dos casos: si el árbol esta vacio o si el árbol NO esta vacio, en ese caso tengo que recorrer desde la raiz hasta el lugar correspondiente. Se inserta en las hojas

```
Procedure preOrden ( a : arbol );
begin
  if ( a<> nil ) then begin
   write (a^.dato);
   preOrden (a^.HI);
   preOrden (a^.HD);
  end;
end;
```

3 Imprimir ABB sin orden

```
Procedure crear (var A:árbol; num:integer);
Begin
 if (A =nil) then
 begin
   new(A);
   A^.dato:= num; A^.HI:= nil; A^.HD:= nil;
 end
 else
  if (num < A^.dato) then crear(A^HI,num)
  else crear(A^.HD,num)
End;
```

2 Carga de elementos al arbol

- Si quiero imprimir de forma decreciente, primero voy hacia derecho a izquierdo
- Si quiero de forma creciente, empiezo de izquierda a derecha


```
Procedure enOrden ( a : arbol );
begin
   if ( a<> nil ) then begin
    enOrden (a^.HI);
    write (a^.dato);
    enOrden (a^.HD);
   end;
end;
```

4 Imprimir para árbol ordenado

BUSQUEDA DE VALOR EN ARBOL

Esta operación retorna un puntero al nodo en el que el árbol que tiene valor X o NIL si no existe. Hay una solución recursiva e iterativa. La versión iterativa devuelve un valor booleano que indica si el dato se encuentra o no en el árbol.

```
function Buscar (a:arbol; x:elemento): arbol;
begin
 if (a=nil) then Buscar:=nil
 else if (x= a^.dato) then Buscar:=a
      else
       if (x < a^.dato) then Buscar:=Buscar(a^.hi ,x)
       else  Buscar:=Buscar(a^.hd ,x)
```

5 Modo recursivo

Merge

La operación merge consiste en generar una nueva estructura de datos (arreglo o lista), ordenada a partir de la mezcla de 2 o más estructuras de datos previamente ordenadas.

Las estructuras que se combinan guardan el mismo orden lógico interno.

Ejemplo: supongamos que se dispone de dos listas con los nombres de libros ordenados alfabéticamente y se pide generar una única lista de libros ordenada alfabéticamente

Libro-> elemento de la lista

Estante-> lista

Biblioteca-> 2 listas

```
program merge;
Type
 lista =^nodo;
 nodo = record
  dato:string;
  sig:lista;
end;
```

```
end
else (e1 <> nil) and (e2 = nil) then begin
  min:= e1^.dato;
  e1:= e1 ^.sig;
end
else (e1 = nil) and (e2 <> nil) then begin
```

TALLER DE PROGRAMACION RESUMEN

MERGE ACUMULADOR

```
program merge;
cantE 2;
type
lista =^nodo;
nodo = record
 impuesto:string;
 monto:real
 sig:lista;
end;
estantes = array[1..cantE] of lista;
Procedure minimo(var todos:estantes; var nomM:string; var mon:real);
var
indiceMin,i:integer;
begin
 nomM:= 'ZZZ'; indiceMin:= -1;
 for i:= 1 to cantE do
  if (todos[i] <> nil) then
   if (todos[i]^.impuesto <= nom) then indiceMin:= i;
  if (indiceMin <> -1) then
   begin
    nomM:= todos[indiceMin]^.impuesto;
    mon:= todos[indiceMin]^.monto;
    todos[indiceMin]:= todos[indiceMin]^sig;
   end;
end;
Procedure merge (todos:estantes; var Enuevo:lista);
Begin
 Enuevo:= nil;
 minimo (todos,minNombre, monto);
 while (minNombre <> 'ZZZ') do
  begin
   actual:= minNombre; montoTotal:=0;
   while ((minNombre <> 'ZZZ') and (minNombre = actual) )do
   begin
    montoTotal:= montoTotal + monto;
    minimo (todos,minNombre,monto);
   end;
   agregarAtras(Enuevo,minNombre,montoTotal);
  end;
End;
Var
 todos: estantes;
 totalMontos: lista;
Begin
 generarEstantes (todos);
 merge (todos,totalMonto);
End.
```


TALLER DE PROGRAMACION- RESUMEN

OTRA SOLUCION

```
{PROCEDURE MINIMO}
procedure minimo(//; var minX://; var monto:real);
var
   indiceMin,i:integer;
begin
   minX:="ZZZ",
   indiceMin:=-1;
   for i:=1 to // do begin
     if //(i)<> nil then begin
       if //(i)^/// <= minX then begin
          indiceMin:= i;
          minX:=///i^..,
          monto:=///i^..;
       end;
     end;
   end;
   if indiceMin <> -1 then
      ///[indiceMin]:=///[indiceMin]^.sig;
end;
```

```
{MERGE ACUMULADOR}
procedure merge(todos:estantesl var Enuevo:lista)
var
   ult:///;
   montoTotal monto,minX,actual:///;
begin
   ult:=nil;
   minimo(///,minX,monto);
   while minX <> // do begin
      actual:=minX;
      montoTotal:=0;
      while ((minX <> //) and (minX = actual) do begin
         montoTotal:=montoTotal+monto;
         minimo(//,minX,monto);
      end;
      agregarAtras(///,//,actual,montoTotal);
end;
```

MÓDULO OBJETOS

Paradigma más utilizado. Al contrario de la programación procedimental que enfatiza en los algoritmos, la POO enfatiza en los datos.

Los paradigmas de programación indican la manera de estructurar y organizar las tareas de nuestro programa. Los lenguajes suelen ser multiparadigma.

Clase: abstracción de los atributos (características), operaciones, relaciones y semántica comunes a un conjunto de objetos. Así, una clase representa al conjunto de objetos que comparten una estructura y un comportamiento comunes. Cada clase tiene sus propias características y comportamientos. Una clase contiene muchos objetos.

En el diseño de un POO contiene al menos los siguientes pasos:

1. Identificar los objetos del sistema
2. Agrupar en clases a todos objetos que tengan características y comportamientos comunes
3. Identificar los datos y operaciones de cada una de las clases
4. Identificar las relaciones que pueden existir entre las clases

Ejemplo: una clase puede describir las propiedades genéricas de un ejecutivo de una empresa y un objeto representará a un ejecutivo especifico.

Objeto abstracción de un objeto del mundo real, definiendo qué lo caracteriza (estado interno) y qué acciones sabe realizar (comportamientos). Elemento individual con su propia identidad.

Sus características principales es la *abstracción, encapsulamiento de datos, ocultación de datos, herencia* y *polimorfismo*. El estado de un objeto viene determinado por los valores que toman sus datos.

Abstracción: propiedad que consiste en tener en cuenta solo los aspectos más importantes desde un punto de vista determinado y no tener en cuenta los restantes aspectos.

Estado interno: compuesto por datos/ atributos que caracterizan al objeto y relaciones con otros objetos con los cuales colabora. Se implementan a través de *variables de instancia*. Si se desea leer datos de un objeto, se llama a una función miembro del objeto. Se accede a los datos y se devuelve un valor. No se puede acceder a los datos directamente ya que están ocultos.

Comportamiento: acciones o servicios a los que sabe responder el objeto. Se implementan a través de métodos de instancia que operan sobre el estado interno. Los servicios que ofrece al exterior constituyen la interfaz.

Encapsulamiento (ocultamiento de información): se oculta la implementación del objeto hacia el exterior. Desde el exterior solo se conoce la interfaz del objeto. Facilidad el mantenimiento y evolución del sistema, ya que no hay dependencias entre las partes del mismo. Utilizado para protegerlos de alteraciones accidentales. Si se desea modificar los datos de un objeto, se conoce exactamente cuáles son las funciones que interactúan con miembros del objeto.

MENSAJE

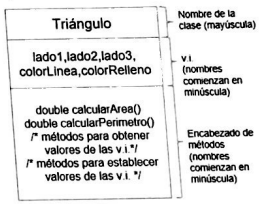
Provoca la ejecución del método indicado por el nombre del mensaje, puede llevar datos (parámetros del método) y puede devolver un dato (resultado del método).

CLASE

Describe el conjunto de objetos comunes (mismo tipo). El cual consta de: declaración de las variables de instancia que implementaran el estado del objeto, la codificación de los métodos que implementan su comportamiento.

Un objeto se crea a partir de una clase (el objeto es instancia de una clase)

Representación gráfica de una clase



INSTANCIACIÓN (creación del objeto)

Es la realización de los objetos descritos en una clase. Constan de datos atributos descritos en la clase y se pueden manipular con operaciones definidas dentro de ellas.

Se realiza enviando un mensaje de creación a la clase. En la cual se reserva espacio para el objeto, ejecuta el código inicializador o constructor (puede tomar valores pasados en el mensaje de creación. Inicializa el objeto con valores recibidos).

Devuelve la referencia al objeto. Asocia la referencia a una variable (a través de ella podemos enviarle mensajes al objeto)

PROGRAMA ORIENTADO A OBJETOS

Los programas se organizan como una colección de objetos que cooperan entre si enviándose mensajes. Cada objeto es instancia de una clase. Los objetos se crean a medida que se necesitan.

El usuario le envía un mensaje a un objeto, en caso de que un objeto conozca a otro puede enviarle mensaje, así estos fluyen por el sistema.

Cuando los objetos no son necesarios, se borran de memoria.

DESARROLLO

Identificar los objetos a abstraer en nuestra aplicación, identificar las características y acciones relevantes que hacen.

Los objetos con características y comportamiento similar serán instancia de una misma clase.

INSTANCIACIÓN

- Declarar variable para mantener la referencia *NombreDeClase miVariable;*

- Enviar a la clase el mensaje de creación y guardar la referencia *miVariable= new NombreDeClase(valores para la inicialización);*
- Se puede unir los dos pasos anteriores *NombreDeCLase miVariable= new NombreDeClase(...);*
- Secuencia de pasos en la instanciación: reserva de memoria (las variables de instancia se inicializan a valores por defecto o explícito, si hubiese), ejecución del constructor (código para inicializar variables de instancia con los valores que enviamos en el mensaje de creación), asignación de la referencia a la variable.

Introducción a Java

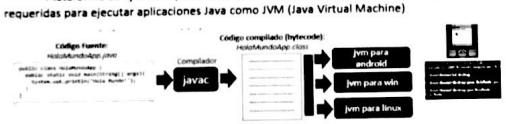
Es un lenguaje de propósito general con paradigmas imperativo y orientado a objetos.

Permite generar aplicaciones multiplataforma.

Plataforma Java:

Plataforma de desarrollo (JDF: Java Developer Kit) que incluye compilador, depurador y generador de documentación

Plataforma de ejecución(JRE: Java Runtime Environment) que incluye componentes requeridas para ejecutar aplicaciones Java como JVM (Java Virtual Machine)



PROGRAMA PRINCIPAL

```
public class NombreAplicacion {
   public static void main(String[] args) {
       /* Código */
   }
}
```

- *Main=* "programa principal" () delimita el cuerpo
- Sentencias de código separadas por ;
- Se recomienda indentar el código para facilitar su lectura
- Comentarios: lineas múltiples o linea única
- Es case-sensitive (sensible a las mayúsculas y minúsculas)

DECLARACION VARIABLES LOCALES A METODO (main u otro)

- Se declaran en zona de código y no toman valor por defecto: *Tipo nombreVariable*
- Convención de nombres: comenzar con minúscula, luego cada palabra en mayúscula
- Asignación: nombreVariable= valor;
- Tipo primitivos: la variable almacena un valor

Tipo primitivo	Ejemplo

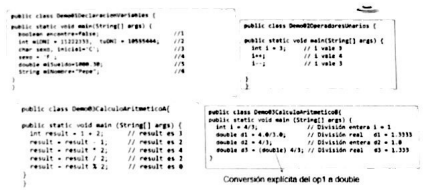
boolean	True o false
char	'a' 'e'
int	102
double	123.4

- *String* para manipular cadenas

OPERADORES

Aritmeticos	Unarios	Relacionales	Condicionales	Concatenación
+ suma	++ incrementa 1	== igual	&& AND	+ de string
- resta	-- decremento 1	!= distinto	|| OR	
* multiplicación		> mayor	! NOT	
/ división		>= mayor o igual		
% operador resto		<= menor igual		
		< menor		

EJEMPLOS DE DECLARACION DE VARIABLES



MOSTRAR DATOS EN LA SALIDA ESTANDAR

- System.out.print(...)
- System.out.println(...)

Para mostrar varios datos uso + de concatenación: System.out.println("Hola mundo"+ año+ "!)

PAQUETES IMPORTADOS

Importo el Paquete de lectura, Lector o GeneradorAleatorio

ESTRUCTURAS DE CONTROL

Selección

```
if (condición)
   acción(es) a realizar cuando
   condición es true
else
   acción(es) a realizar cuando
   condición es false
```

Encerrar entre {} en caso de incluir varias sentencias.
Cuando sólo incluye una sentencia, finalizarla con ;

Leer acerca del case (switch en java) en:
http://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html

Iteración pre-condicional

```
while (condición)
   acción(es) a realizar cuando
   condición es true
```

Iteración post-condicional

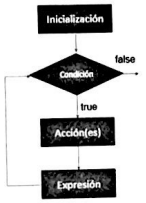
```
do{
   acción(es)
} while (condición)
```

Diferencia do-while y while
- Ejecuta acción(es) y luego evalúa condición
- Cuando condición es true => ejecuta otra vez acción(es)
- Cuando condición es false => finaliza do

6 Selección, pre condicional, post condicional

Repetición for (inicialización; condición; expresión)
acción(es)

- *Inicialización*: expresión que se ejecuta una vez al comienzo y da valor inicial a la variable índice.
- *Condición*: expresión lógica, se evalúa antes de comenzar una nueva iteración del for; cuando da false termina el for.
- *Expresión*: expresión que se ejecuta al finalizar cada iteración del for (incr. o decr. del índice).



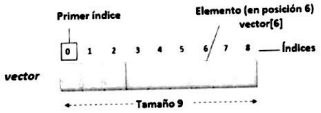

```
int i;
for (i=1; i<= 10;  i++)
    System.out.println(i);
```
¿Qué imprime?
¿Modificar para imprimir pares?

```
int i;
for (i=10; i > 0;  i=i-1)
    System.out.println(i);
```
¿Qué imprime?
¿Es lo mismo poner i-- ?

7 Iterativa

ARREGLOS UNIDIMENSIONALES

Almacenan un número fijo de valores primitivos u objetos del mismo tipo. Acceso de forma directa a las posiciones. La dimF se establece al crearlo. En INDICE es entero, comenzando desde 0.



Arreglos unidimensionales - Vector

- Declaración
TipoElemento [] nombreVariable;
- Creación
nombreVariable = new TipoElemento[DIMF];
- Acceso a elemento

Ejemplo:

```
int [] contador = new int[10];
for (i=0;i<10;i++) contador[i]=i;

System.out.println("La Pos. 1 tiene " +contador[1]);
```


Una matriz es un arreglo con 2 dimensiones

ARREGLOS BIDIMENSIONALES- MATRICES → p 167

Ejemplos de uso: sala de un teatro, cartón de bingo, etc.

- Declaración
TipoElemento [][] nombreVariable;
- Creación
nombreVariable = new *TipoElemento [DIMF][DIMC]*;
- Acceso a elemento
nombreVariable [posFil] [posCol]
- Ejemplo:

```
int [][] tabla = new int[3][4];
int i, j;
for (i=0;i<3;i++)
  for (j=0;j<4;j++)
    tabla[i][j]=GeneradorAleatorio.generarInt(10);
System.out.println("La Pos. 1,2 tiene " +tabla[1][2]);
```

Gráficamente


Pensar las operaciones:
- *Imprimir el contenido de la matriz*
- *Imprimir el contenido de una columna específica*
- *Sumar los elementos de una fila específica*

Definición de clases

SINTAXIS

```
public class NombreDeClase {
    /* Declaración del estado del objeto*/
    /* Declaración de constructor(es) */
    /* Declaración de métodos que implementan acciones */
}
```

DECLARACION DE ESTADO

Estado interno:

- Datos de tipos primitivos *TipoPrimitivo nombreDato;*
- Referencias a otros objetos *NombreDeClase nombreDato;*

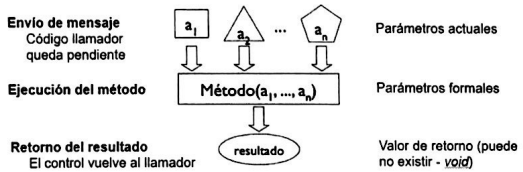
Debemos anteponer a la declaración la palabra private para lograr encapsulamiento (ocultamiento de la información) Las v.i.s privadas pueden ser accedidas solo dentro de la clase que las declara. ***private*** *double precio;*

En la declaración del dato de puede dar un valor inicial ***private*** *double precio=10,5;*

DECLARACION DEL COMPORTAMIENTO

```
public TipoRetorno nombreMetodo ( lista de parámetros formales ) {
    /* Declaración de variables locales al método */
    /* Cuerpo del método */
}
```

- **Public** indica que el método forma parte de la interfaz
- **TipoRetorno:** tipo de dato primitivo/nombre de clase/ void (si no retorna dato)
- **nombreMetodo:** verbo seguido de palabras por convención de nombres
- **Lista de parámetros:** datos de tipos primitivos u objetos. Separación por coma entre parámetros, solamente pasaje de valores.
- **Declaración de variables locales**
- **Cuerpo:** código puedo utilizar estado y modificarlo o devolver resultado mediante ***return***



DECLARACION DE COMPORTAMEINTOS

Parametros únicamente por valor

a) Parámetro dato primitivo: el parámetro formar recibe copia del valor del parámetro actual. Si modifico parámetro formal, no altera parámetro actual

```
Main
  Libro l1 = new Libro();

  int x = 1;
  l1.hacerUno(x);
  System.out.println(x); ¿Qué imprime?
```

```
public class Libro{
  ...
  public void hacerUno(int y){
     y++;
  }
}
```

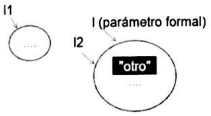
Imprime: 1

b) Parámetro objeto: parámetro formal recibe copia de la referencia del parámetro actual. Si se modifica el estado interno del objeto del parámetro formal, el cambio en el estado es visible en el parámetro actual.

```
Main
  Libro l1 = new Libro();
  Libro l2 = new Libro();
  l2.setTitulo("Java");

  l1.hacerDos(l2);
  System.out.println(l2.getTitulo()); ¿Qué imprime?
```

```
public class Libro{
  ...
  public void hacerDos(Libro l){
     l.setTitulo("otro");
  }
}
```



Imprime: "otro"

INSTANCIACION

- Declarar variable para mantener la referencia NombreDeClase miVariable;
- Enviar a la clase el mensaje de creación miVariable= new NombreDeClase();
- Se pueden unir los pasos anteriores
  NombreDeClase miVariable= new NombreDeClase;

Secuencia de pasos en la creación: *reserva de memoria*, las variables de instancia se inicializan a valores por defecto o explícito (si hubiese)- *ejecución del constructor* (código para inicializar variables de instancia con los valores que enviamos en el mensaje de creación)- *asignación de la referencia a la variable*

DECLARACION DE CONSTRUCTORES

Se ejecuta tras alocar el objeto e inicializar las v.i. Su objetivo es inicializar las v.i

Si la clase no declara ningún constructor, Java incluye uno nulo. Puede haber varios constructores, Java identifica cual se invoca por el número y tipo de parámetros.'

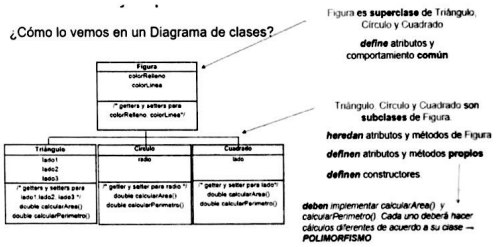
CONCEPTO DE HERENCIA

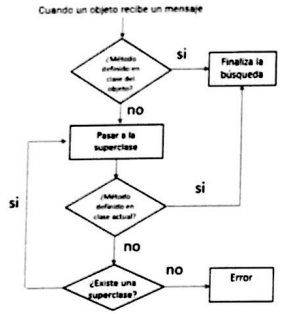
Mecanismo que permite que una clase herede características y comportamientos de otra clase. A su vez, la clase hija define sus propias características y comportamiento. Su ventaja es la reutilización de código. Clases diferentes se pueden conectar unas con otras de modo jerárquico.

Ejemplo: la súper clase figura y las clases Triangulo, Circulo y Cuadrado heredarían de esta.

POLIMORFISMO

Propiedad de que un operador o una función actúen de modo diferente en función del objeto sobre el que se aplican. Aquella en que una operación tiene el mismo nombre en diferentes clases, pero se ejecuta de diferentes formas en cada clase.





8 Búsqueda de método en jerarquía de clases

La herencia se define mediante la parable ***extends.***

Los atributos declarados en una superclase como privados, no son accesibles en subclases, para accederlos en una subclase debo implementar getters y setters.

CLASES Y METODOS ABSTRACTOS

En algunas situaciones podemos encontrarnos con superclases de las cuales no nos resulta útil definir un nuevo objeto, sino que los nuevos objetos se van a instanciar desde sus subclases. La clase Figura sería una clase abstracta

Una clase abstracta es una clase que no puede ser instanciada. Define características y comportamientos común para un conjunto de clases (subclases). Puede definir métodos abstractos que deben ser implementados por las subclases.

Son métodos para los cuales se define su encabezado, pero no se los implementa. Las clases que hereden de una superclase abstracta deberán implementar sus métodos abstractos.

Ejemplo: para la clase abstracta Figura podemos declarar los métodos calcularArea y calcularPerimetro como métodos abstractos. Y las clases Triangulo, Cuadrado y Circulo deberán implementar estos métodos.

Declaración de clase abstracta: *anteponer **abstract** a la palabra class*

```
public abstract class NombreClase {
      /* Definir atributos */
      /* Definir métodos no abstractos (con implementación) */
      /* Definir métodos abstractos  (sin implementación) */
}
```

Declaración de método abstracto: *encabezado del método (sin código) anteponiendo **abstract** al tipo de retorno*

```
public abstract TipoRetorno nombreMetodo(lista parámetros formales);
```

Ejemplo:

```
public abstract class Figura{
    ...
    public abstract double calcularArea();
    public abstract double calcularPerimetro();
}
```


Superclase

```
public abstract class Figura{
    private String colorRelleno, colorLinea;

    public String getColorRelleno(){
        return colorRelleno;
    }
    public void setColorRelleno(String unColor){
        colorRelleno = unColor;
    }
    ...
    public abstract double calcularArea();
    public abstract double calcularPerimetro();
}
```

MÉTODOS ABSTRACTOS

Subclase

```
public class Cuadrado extends Figura{
    private double lado;

    /*Constructores*/
    public Cuadrado(double unLado,
                    String unColorR,
                    String unColorL){
        lado = unLado;
        setColorRelleno(unColorR);
        setColorLinea(unColorL);
    }
}
```

El objeto que está ejecutando (this) se envía un mensaje a sí mismo

setColorRelleno(unColorR)
ó this.setColorRelleno(unColorR)

Recordar ¿Cómo se busca el método a ejecutar?

Implementemos los métodos abstractos

Superclase

```
public abstract class Figura{
    private String colorRelleno, colorLinea;

    public Figura(String unCR, String unCL){
        setColorRelleno(unCR);
        setColorLinea(unCL);
    }
    public String toString(){
        String aux = "CR:" + getColorRelleno() +
                     "CL:" + getColorLinea();
        return aux;
    }
    public String getColorRelleno(){
        return colorRelleno;
    }
    public void setColorRelleno(String unColor){
        colorRelleno = unColor;
    }
    ...
    public abstract double calcularArea();
    public abstract double calcularPerimetro();
}
```

Agregamos a la superclase un constructor y un toString

Subclase

```
public class Cuadrado
    private double lado;

    /*Constructores*/
    public Cuadrado(do
unColorR,
                    String
unColorL){
        super(unColorR,unColorL);
        setLado(unLado);
    }

    /* Metodos getLado y setLado  */

    /* Métodos calcularArea y calcularPerimetro */

    public String
        String aux =        ????          +
                    "Lado:" + getLado();
        return aux;
```

super(...)
Invoco al constructor de la superclase
Al declarar un constructor en la superclase esta invocación debe *ir como primera línea*

super.toString()

super es la referencia al objeto que está ejecutando
super.toString() → El objeto se envía un mensaje a sí mismo
La búsqueda del método inicia en la clase superior a la actual.

Superclase

```
public class Figura{
    private String colorRelleno, colorLinea;

    public Figura(String unCR, String unCL){
        setColorRelleno(unCR);
        setColorLinea(unCL);
    }
    public String toString(){
        String aux = "Area:"+this.calcularArea()
                   + "CR:" + getColorRelleno()
                   + "CL:" + getColorLinea();
        return aux;
    }
```

Subclases

```
public class Cuadrado extends
Figura{
    private double lado;

    /*Constructores*/
    public Cuadrado(double
unLado,
                    String
unColorR,
                    String
unColorL){
        super(unColorR,unColorL);
        setLado(unLado);
    }

    /* Metodos getLado y
setLado*/

    /* Métodos calcularArea y
       calcularPerimetro */

    public String toString(){
        String aux =
super.toString()+
                    "Lado:" +
...
```

```
public class Circulo extends
Figura{
    private double lado;

    /*Constructores*/
    public Cuadrado(double
unRadio,
                    String
unColorR,
                    String
unColorL){
        super(unColorR,unColorL);
        setRadio(unRadio);
    }

    /* Metodos getRadio y
setRadio*/

    /* Métodos calcularArea y
       calcularPerimetro */

    public String toString(){
        String aux =
super.toString()+
                    "Radio:" +
getRadio();
        return aux;
    }
}
```

Polimorfismo: objetos de clases distintas responden al mismo mensaje de distinta forma.

Binding dinámico: se determina en tiempo de ejecución el método a ejecutar para responder a un mensaje.

Ventaja: Código genérico, reusable.

```
Cuadrado(10,"rojo","negro");
o(c.toString());
Circulo(5,"verde","azul");
System.out.println(c2.toString());
```

RESUMEN

Encapsulamiento: permite construir componentes autónomos de software, es decir independientes de los demás componentes. La independencia se logra ocultando detalles internos (implementación) de cada componente. Una vez encapsulado, el componente se puede ver como una caja negra de la cual sólo se conoce su interfaz.

Herencia: permite definir una nueva clase en términos de una clase existente.

La nueva clase hereda automáticamente todos los atributos y métodos de la clase existente, y a su vez puede definir atributos y métodos propios.

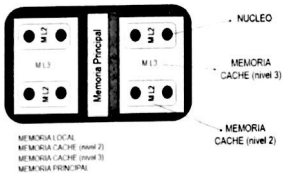
Polimorfismo: objetos de clases distintas pueden responder a mensajes con nombre (selector) sintácticamente idénticos. Esto permite realizar código genérico, altamente reusable.

Entre los beneficios de la POO, podemos mencionar producir SW que sea:

- ***Natural***. El programa queda expresado usando términos del problema a resolver, haciendo que sea más fácil de comprender.
- ***Fiable***. La POO facilita la etapa de prueba del SW. Cada clase se puede probar y validar independientemente.
- ***Reusable***. Las clases implementadas pueden reusarse en distintos programas. Además, gracias a la herencia podemos reutilizar el código de una clase para generar una nueva clase. El polimorfismo también ayuda a crear código más genérico.
- ***Fácil de mantener***. Para corregir un problema, nos limitamos a corregirlo en un único lugar.

MODULO CONCURRENTE

EVOLUCION DE LAS ARQUITECTURAS



Para poder explotar este hardware, los programas deben ser concurrentes.

La memoria principal la comparten todos los núcleos.

Memoria local para un núcleo, memoria nivel 2 para un par, memoria nivel 3 para 2 pares.

Un programa concurrente se divide en tareas, las cuales se ejecutan al mismo tiempo y realizan acciones para cumplir un objetivo común. Estos pueden compartir recursos, coordinarse y cooperar. Sus características clases con la *comunicación* y la *sincronización* (se volvieron un concepto clave en las Cs. De la computación, influye tanto en hardware como software)

Ejemplo: los automóviles son los procesos ejecutados, carriles: múltiples procesadores. Los automóviles deben sincronizarse para no chocar.

En la programación concurrente la comunicación es mediante envió de mensajes y memoria compartida, donde

- Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos
- También el lenguaje debe proveer el protocolo adecuado
- Para que la comunicación sea efectiva los procesos deben saber cuándo tienen mensajes para leer y cuando deben transmitir mensajes
- Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella
- Lógicamente no pueden operar simultáneamente, lo que obliga a bloquear y liberar el acceso a memoria.
- La solución más elemental es una variable de control que habilite o no el acceso de un proceso a la memoria compartida

AMBIENTE CMRE

Se permite declarar más de un robot, las áreas pueden ser privadas, compartidas o parcialmente compartidas. La comunicación es mediante *enviar* o *recibir* mensaje. Y la sincronización mediante *bloquear* o *liberar* esquina

Áreas

- areaC: cualquier robot puede circular por la misma. Gris. Memoria principal
- areaP: solo puede haber en ella un único robot. Azul. Memoria local